[**Linux编程基础——GDB（查看数据）**](http://www.cnblogs.com/TianFang/archive/2013/01/21/2869474.html)

<http://www.cnblogs.com/TianFang/archive/2013/01/21/2869474.html>

# [Linux编程基础——GDB（查看数据）](http://www.cnblogs.com/TianFang/archive/2013/01/21/2869474.html)

**查看栈信息**

当程序被停住了，首先要确认的就是程序是在哪儿被断住的。这个一般是通过查看调用栈信息来看的。在gdb中，查看调用栈的命令是backtrace，可以简写为bt。

    (gdb) bt  
    #0 pop () at stack.c:10  
    #1 0x080484a6 in main () at main.c:12

也可以通过info stack命令实现类似的功能（我更喜欢这个命令）：

    (gdb) info stack  
    #0 pop () at stack.c:10  
    #1 0x080484a6 in main () at main.c:12

**查看源程序**

当程序断住是，gdb会显示当前断点的位置：

    Breakpoint 1, pop () at stack.c:10  
    10 return stack[top--];

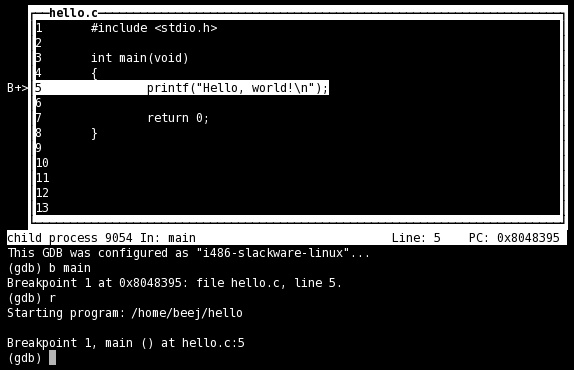
可以用list命令来查看当前断点附近的程序的源代码：

    (gdb) list  
    5 int top = -1;  
    6  
    7  
    8 char pop(void)  
    9 {  
    10 return stack[top--];  
    11 }  
    12  
    13 void push(char c)  
    14 {

list命令后面还可以更一些参数，来显示更多功能：

* <linenum>    行号。
* <+> [offset]    当前行号的正偏移量。
* <-> [offset]     当前行号的负偏移量。
* <filename:linenum> 文件的中的行行。
* <function>             函数的代码
* <filename:function>     文件中的函数。
* <\*address> 程序运行时的语句在内存中的地址。

不过，就算有这些信息，查看代码仍然不大方便。现在新版的gdb都带一个tui的功能，可以通过focus命令开启，其主要界面如下：



这个界面比起list来说方便多了，能高亮当前语句的执行位置，步进时也会跟着变化，有点使用Turbo C的感觉。

不知道是不是由于focus比较新的缘故，貌似网上并没有多少文章介绍它，虽然它比较容易上手，但也有不少可以介绍的地方，限于篇幅我这里就不做更多的说明，感兴趣的朋友可以看下[gdb的gui用法](http://www.cnblogs.com/xuxm2007/archive/2011/08/13/2137055.html)这篇文章。

**查看运行时数据**

gdb中查看变量的命令是print，一般用它的简写形式p。它的语法如下：

    print [</format>] <expr>

其中参数expr可以是一个变量，也可以是表达式。format表示输出格式，例如，可以用/x来将结果按16进制输出。如下是几个基本的例子：

    (gdb) p top  
    $16 = 1  
    (gdb) p &top  
    $17 = (int \*) 0x804a014 <top>  
    (gdb) p 3+2\*5  
    $18 = 13  
    (gdb) p /x 3+2\*5  
    $19 = 0xd

format的取值范围有如下几种：

* x 按十六进制格式显示变量。
* d 按十进制格式显示变量。
* u 按十六进制格式显示无符号整型。
* o 按八进制格式显示变量。
* t 按二进制格式显示变量。
* a 按十六进制格式显示变量。
* c 按字符格式显示变量。
* f 按浮点数格式显示变量。

**查看函数返回值**

查看函数返回值是在调试的过程中经常遇到的需求。例如，对于如下函数

    int foo()  
    {  
        return 100;  
    }

我们可以以如下方式获取函数的返回值：

1. 通过finish命令运行至函数结束，此时会打印函数返回值。

    (gdb) finish  
    Run till exit from #0 foo () at main.c:9  
    main () at main.c:15  
    15 }  
    Value returned is $2 = 100

2. 返回值会存储在eax寄存器中，通过查看信息可以获取返回值。

    (gdb) p $eax  
    $3 = 100  
  
    (gdb) info registers   
    eax 0x64 100

**查看连续内存**

可以使用GDB的"@"操作符查看连续内存，"@"的左边是第一个内存的地址的值，"@"的右边则你你想查看内存的长度。

例如，对于如下代码：int arr[] = {2, 4, 6, 8, 10};，可以通过如下命令查看arr前三个单元的数据。

    (gdb) p \*arr@3  
    $2 = {2, 4, 6}

**查看内存**

可以使用examine命令（简写为x）来查看内存地址中的值。x命令的语法如下所示：

    x /<n/f/u> <addr>

* n 表示显示内存的长度，也就是说从当前地址向后显示几个地址的内容。
* f 表示显示的格式，如果是字符串，则用s，如果是数字，则可以用i。
* u 表示从当前地址往后请求的字节数，默认是4个bytes。（b单字节，h双字节，w四字节，g八字节）
* <addr> 表示一个内存地址。

例如：以两字节为单位显示前面的那个数组的地址后32字节内存信息如下.

    (gdb) x /16uh arr  
    0xbffff4cc: 2 0 4 0 6 0 8 0  
    0xbffff4dc: 10 0 34032 2052 0 0 0 0

**自动显示**

在VisualStudio中，可以通过监视窗口动态查看变量的值。在gdb中，也提供了类似的命令display，它的语法是：

    display <expr>  
    display /<fmt> <expr>  
    display /<fmt> <addr>

expr是一个表达式，fmt表示显示的格式，addr表示内存地址。当你用display设定好了一个或多个表达式后，只要你的程序被停下来（单步跟踪时），GDB会自动显示你所设置的这些表达式的值。

几个相关的命令如下：

* undisplay <dnums...>        不显示dispaly
* delete display [dnums]    删除自动显示，不带dnums参数则删除所有自动显示，也支持范围删除，如： delete display 1,3-5
* disable display <dnums...>    使display失效
* enable display <dnums...>    恢复display
* info display        查看display信息